Reconfigurable Network Processing: The FPGA case

Carlos Zerbini[1] and Jorge M. Finochietto[2]

[1] Universidad Tecnológica Nacional, Córdoba
[2] Universidad Nacional de Córdoba - CONICET, Córdoba
czerbini@electronica.frc.utn.edu.ar,jfinochietto@efn.uncor.edu

Abstract. As communication networks evolve towards 100 gigabit per second rates to address an increasing demand of data traffic, network processing solutions must be revised and upgraded to support this need. Meanwhile, Field Programmable Gate Array (FPGA) technology is becoming a much more interesting platform where to integrate network processing capabilities and compete with current available solutions. In this paper, we argue that FPGAs can play a significant role in this area. To this end, a general discussion on the technology is first introduced to later focus on the specific requirements to implement network processing architectures. Finally, based on our previous experience on building network devices on FPGAs, we discuss a case study to illustrate some of the main drivers to consider FPGA as a interesting solution for network processing.

Keywords: Packet networks, network processing, FPGA, high-speed communications

1 Introduction

Communication networks capacities are expected to evolve towards several hundred of gigabit per second (Gbps) rates in coming years [1]. New, bandwidth-demanding applications and broad adoption of media streaming are leading drivers of this tendency [2]. Consolidation of services over Ethernet [3] and its application as Local, Metropolitan and Wide Area Network also play an important role in it. Even though transmission links can offer high capacity, congestion arises due to network processing bottlenecks.

Data granularity is the main preventer for network processing scalability. Today's networks carry data units of variable-length (i.e., packets), which are independently processed and routed at each hop. Besides, networks must support 64-byte packets, which at 100 Gbps bounds the packet processing time to about 5 nanoseconds (ns). Typically, no throughput is guaranteed in such a transfer, only best-effort service is provided.

Several factors are driving the need for new, high-performance network processing architectures. A clear understanding of their relations, tendencies and suitable technologies is necessary in order to achieve innovative and scalable solutions. In this context, Field Programmable Gate Array (FPGA) technology is becoming an interesting arena where to experiment and implement these solutions. It is expected that FPGAs will compete with former solutions like Network Processors (NPs) and Application-Specific Integrated Circuits (ASICs). Besides high performance, FPGAs can offer much flexibility to adapt to new protocols and running configurations.

Based on the main functional aspects of network processing, this paper discusses and analyzes the use of FPGA technology. Previous work in this area is surveyed in Section 2. Section 3 introduces available technologies and compares their flexibility and performance. Key processing functions and their integration in reconfigurable platforms are discussed in Section 4. A case study based on our experience in the area is presented in Section 5. Finally, Section 6 concludes the work.

2 Related Work

Much research has been carried out on programmable architectures for processing network traffic. Both software and hardware alternatives have emerged with specific and sometimes complementary features.

A well-known platform based on commodity off-the-shelf hardware is the Click [4] modular router. It consists of a library of software objects, which can be connected by a simple own scripting language. Click can be run as a kernel process for enhanced performance; however, it suffers of limitations inherent of software-based approaches. Cusp [5] and Nikander *et. al.* [6] consider mapping Click to hardware; however, little or preliminar information exist about the performance of these approaches and their feasibility.

Early work on FPGA-based networking platforms was made by Brebner [7], where both logic and a PowerPC core were used to implement a mixed IPv4/IPv6 router called MIR. This work demonstrated the feasibility and advantages of such an implementation. Later advances on this area are RiceNIC [8] and NetFPGA [9]. The latter has became very popular in teaching and research based on a provided hardware pipeline.

Inspired by these ideas, in a previous work [10] we developed atomic hardware units for packet processing in a context where packets are handled as flows to mitigate data granularity issues. This approach is inline with that proposed by Casado [11] where forwarding decisions are first done in software and later cached in hardware for enhanced speed. The present work aims at comparing technology approaches for network processing, and reviewing lessons learned from our experience in the field.

3 Technology

The main challenges for network processing technology are performance and flexibility. Fastest routers today include ASICs and Custom Silicon for imple-

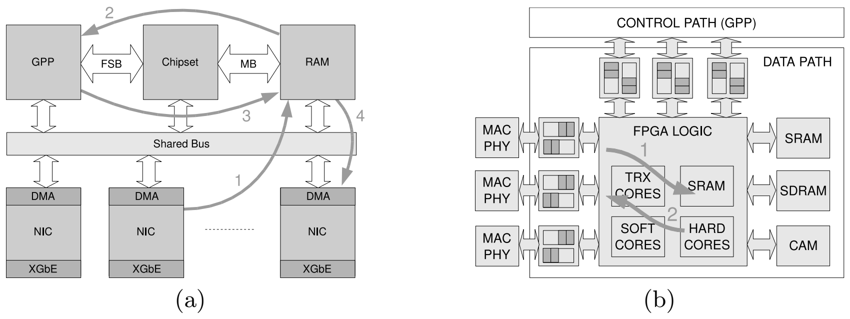

Fig. 1. Typical stages in network processing: (a) GPP-based, (b) FPGA-based

mentation of processing functions. They offer hundreds of parallel logic blocks and state machines to achieve the required throughput, but they typically offer poor or no programmability. In addition, its development cycle is too long and expensive to keep pace with new application's demands.

When flexibility and cost are the main issues, commodity hardware based on General-Purpose Processors (GPPs) running Open Source software is preferred. This is an appealing alternative due to the wide availability of off-the-shelf (e.g., PC) platforms and open software like the Linux operating system, Click (for the data plane), and Xorp/Zebra/Quagga (for control plane). Neverthless, this software-based approach suffers performance and scalability issues when managing high-speed packet flows. Main bottlenecks in multiple-core GPP architectures lay at the Front Side Bus (FSB) connecting GPP to memory and at the GPP itself [12].

Network Processors try to fill the programmability weakness of ASICs, while providing good performance [13]. They are specialized, mostly RISC processor architectures which leverage common properties of network processing functions to implement them in multiple processing elements (PEs). Deep pipelining, specialized instruction sets, network coprocessing units, and multithreading are commonly included. Examples on NPs are Intel IXP family and IBM Power NP. Despite their high performance, processing functions in NPs are difficult to port between different providers. Implementation can be very inefficient if a protocol requires functions not native to NP's instruction set, because they do not allow hardware-level customization. Finally, it is not clear how cost-effective NPs solutions will be for upcomming 100 Gbps networks.

An innovating technology which has became very interesting in the last years is *reconfigurable hardware*, particularly Field Programmable Gate Arrays (FPGAs). As performance gap to ASICs narrows, integration scale and speed increases, and providers include specific-purpose hardware processing blocks, network processing with FPGAs turns into active reseach field. One remarkable feature, unique to FPGAs, is their adaptability to new applications through both *reprogramming* and *reconfiguration*. Combinations of these techniques allow compile- and runtime modifications, tackling a balance between ease of modification and performance. In addition, the extension of the open software paradigm to hardware allows access for open libraries of reusable, well-tested *gateware* (gates + ware).

Figure 3 shows typical network processing stages in GPP and FPGA platforms. In order to achieve a balance between processing power of sequential processors and speed of parallel hardware structures, intelligent FPGA-based Network Interface Cards (NICs) are usually combined with GPP-based commodity hardware, emerging the so-called $hardware - assisted$ approaches. Both slow data path and management functions are performed at low speed by software, while programmable hardware takes care of critical, line-rate processing.

A comparison of all available technologies is summarized in Table 1. In general, FPGAs can offer a good trade-off between performance and programmability.

Table 1. Technology Analysis

	GPPs	FPGAs	ASICs	NPs
Cost	Low	Medium	High	Medium
Parallelism	Limited	High	High	Variable
Programmability	Good	Medium	Low/Null	Medium
Performance	Low	Good	Very Good	Variable
Dev. Cycle	Short	Medium	Long	Medium
Power	Very High	Medium	Low	High
Languages	Standard	Standard/Ad Hoc	Ad Hoc	Standard/Proprietary
Interface	Open	Open/Closed	Proprietary	Proprietary

4 Network Processing

Elemental network processing, i.e. routing, involve fast lookup of paths according to some routing table, forwarding packets to the proper physical output port, and queueing to deal with contention. As protocols and features of Internet evolves, many more processing functions are added to this group. On the one hand, protocols supporting most modern networks are packet based, while architectures should process them at line rate. On the other hand, next-generation network protocols require data plane modifications. That is, not only are packets inter-network routed, but their associated network is also modified according to load, patterns of traffic, packet contents, etc.

Network applications require a common set of functions partitioned as follows:

- *Physical Layer* functions convert physical medium signals into a bitstream with some frame format.

- *Fast Data Path* functions perform operations on all packets, therefore they must operate at line rate.
- *Slow Data Path* functions operate on specific packets. They are typically more complex and slower than Fast Data Path ones.
- *Switch Fabric* stage performs actual forwarding of packets through some kind of interconnection network.
- *Control Path* takes care of configuration and management tasks.

In the following, functions involved in actual applications are presented and their implementation approaches are analized, omiting NPs due to their high heterogeneity. Figure 2 shows their situation in a generic hybrid hardware-software routing scenario.

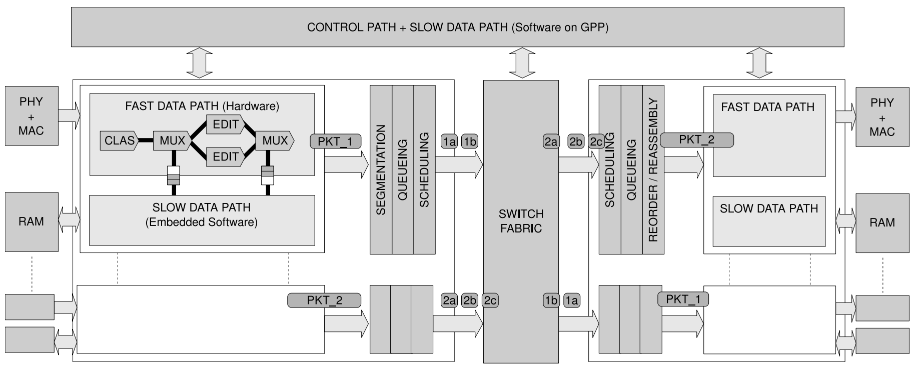

Fig. 2. Generic routing architecture

4.1 Classification

Classifiers are essential blocks of any network processing architecture, and they are of the most resource-demanding ones. Through classification, specific fields of a packet are matched against a set of *rules* in order to decide on the associated processing chain. This decision is mapped to an internal *control word* appended to each data word. Typical applications include Class of Service, traffic billing and filtering.

Multi-field packet classification in hardware basically involves parallel matching in a space of F required fields, which demands memory and hardware area; and subsequent priority encoding of matches, which imposes delay. These are issues to consider as search space grows. Two main approaches deal with this in hardware, namely Ternary Content Addressable Memories (TCAMs) and independent sets partitioning/cross producting. TCAMs offer excellent performance at prohibitively high monetary and power costs, and have in consequence poor scalability [14]. Actual research interest is rather focused on partitioning-based approaches. Recent work reports up to 100 Gbps rates and 50K rules sets [15].

Table 2. Implementation of classification

GPPs	FPGAs	ASICs
Many rules supported in large and slow SDRAM	Dedicated registers; low-latency limited internal SRAM; medium-latency external SDRAM	Limited ad-hoc internal memory; external SDRAM (if interface included)
Intensive algorithms at low speed	Parallel architectures at wire speed	Parallel architectures at wire speed
Fully reprogrammable rules	Fully reprogrammable/ reconfigurable rules	Partially reprogrammable rules

IP Lookup on variable-length fields can be considered a special case of the packet classification problem which poses special technology constraints. In constrast to the *exact* match problem, this scheme is commonly used in routers in order to find the *best* match for a packet field, usually the IP destiny address, in order to decide on the output port for the packet. Nowadays, the use of Classless Interdomain Routing (CIDR) on variable-length addresses requires Longest Prefix Match (LPM) lookup engines, which demand very high performance implementations at actual speeds and extended-length IPv6 addresses. While today's TCAM approach is left aside, algorithmic techniques emerge. Binary search over prefix lengths is a scalable option in software. For hardware implementation, meanwhile, Lulea and Tree Bitmap are the most suitable approaches [16].

4.2 Switching

Switch fabrics play a vital role in switches with multiple physical input/output ports, where contention naturally arises from resource sharing. In IP routers, switch fabrics must have enough intelligence to route packets dynamically through the best output ports for optimal use of available paths. Remarkable implementation features are multicast support (i.e., for multimedia applications), required speedup, internal blocking and cell (ATM)/packet (IP) mode switching.

Time-division switches (TDSs) [17] use either shared-bus or shared-memory approaches. The shared bus, such as PCIe in GPP-based platforms, should offer enough bandwidth to accomodate aggregated traffic from input/output interfaces. Shared memory switches rely on RAM dynamically-allocated queues or external CAM in FPGAs/ASICs, which should support N write/one read accesses per time slot (being N number of I/O ports).

On the other hand, multiple parallel transfers are possible in space-division switches (SDSs), while physical effects, backplane connections, and internal blocking restrict their capacity. This is the preferred architecture for high-performance hardware implementations. Examples of SDSs are Crossbar, Fully Interconnected, Banyan-based, Clos, and Multiplane switches. While ASICs have been widely used for this function, FPGAs are becoming able to integrate switching at speeds in the 10-100 Gigabits-per-second range [18], which is very promising for use in new applications.

Table 3. Implementation of switching

GPPs	FPGAs	ASICs
Shared bus/memory	Time/space division	Time/space division
Low/medium performance	High performance	Very high performance
Little/no optimization possible	Optimization limited by device family	Optimization at design time

Today's communication networks converge to a common, distributed processing platform carrying very heterogeneous services. As a consequence, switching functions must be smart and flexible enough to carry all of them. Different coexisting protocols such as Ethernet, TDM and ATM will require hybrid switch fabrics.

4.3 Queueing

Among switch fabric architecture, buffering strategies define switch performance and technology requirements, and thus its scalability and fields of application. Buffers can be basically applied at input, output, input/output or as part of the switch fabric. Input buffering has less speed demands, but suffers the Head-of-Line (HOL) blocking problem; while output buffering solve this problem but demands N-times line rate writing speed. Virtual Output Queueing (VOQ) at input and combined input/output buffering approaches try to get the best from both worlds, while requiring advanced scheduling algorithms. The Shared-Memory (SM) approach, which was very popular in the last decade, allows flexible allocation of buffer space to different input/output ports, but must support N-times line rate write/read speeds, which drastically limits its scalability. Finally, the Crosspoint-Queueing (XQ) strategy implemented in crossbars allows line rate writing/reading speed and simple scheduling, but requires N^2 independent buffers which degrades buffer sharing capabilities and flexibility to adapt to changing traffic. Hybrid solutions based on this architecture are in study at the present.

As Table 4 shows, GPP centralized memory arquitecture leads to Shared-memory queueing, though its speed requirements are hardly met by this approach. On the other hand, wider options can be implemented in hardware.

Segmentation and Reassembly Segmentation is commonly applied at the interfaces of the switch fabric of in order to facilitate scheduling by doing it at regular intervals, for example in Input-queued swithes. As shown in Figure 2, incoming packets are divided into cells at the input side of the switch, while these

Table 4. Implementation of queueing

GPPs	FPGAs	ASICs
System SDRAM	Internal SRAM or external SDRAM	Internal SRAM or external SDRAM
High latency and capacity	Low latency or high capacity	Low latency or High capacity
Centralized	Centralized/distributed	Centralized

cells are re-assembled into packets after leaving it. This is called $cell - mode$ operation. Drawbacks of this approach are bandwidth-waste from incomplete cells, and additional overhead. Cells from a common packet may be serviced out-of-order, therefore re-ordering is necessary at output, enforcing store-and-forward operation. On the other hand, in $packet - mode$ operation an input/output connection is established for the packet duration, which enables cut-through operation.

4.4 Scheduling

Due to the mentioned contention problem, packet scheduling is always necessary in packet networks. For the general case of fair sharing, fixed-interval (ATM cells) or variable-interval (IP packets) versions of the round robin policy are applied. Advanced policies are used, however, when priorities must be applied to meet Quality-of-Service (QoS) requirements.

Additionally, *policing* and *shaping* functions are usually combined with schedulers in order to meet predefined profiles. While policers drop excess traffic at *inputs* of the switch, shapers commonly delay packets at *outputs* before sending them downstream.

Table 5. Implementation of scheduling

GPPs	FPGAs	ASICs
Programmable	Configurable/ Programmable	Fixed
Slow	Medium	Fast

Table 5 shows that slow and centralized software scheduling is used in GPP-based approaches. Regarding hardware approaches, performance requirements are highly dependent on the adopted queueing strategy. Indeed, Virtual Output Queueing requires optimum matching i.e., the best set of input-output pairs with no conflicts at each slot, which demands execution of complex algorithms at very high speeds. Crosspoint-Queueing, on the other hand, requires simple distributed scheduling but consumes more chip area.

5 A Case Study for Reconfigurable Network Processing

The implementation of network processing architectures in reconfigurable hardware (i.e., FPGA) enables the design of flexible network infrastructures. On the one hand, reconfigurability can provide just *what it is needed* for a given network configuration. Instead of running feature-rich network devices (e.g. commercial routers, firewalls and switches) configured for a specific functionality, reconfigurability can provide just those features required by the actual needs. Thus, the system can become a perfect match between what is needed and what it is implemented on the network device. On the other hand, reconfigurability can offer to implement a network device *where it is needed*. Thus, instead of having a fixed network infrastructure with dedicated devices on specific locations, one physical infrastructure can be shared among multiple logical networks. Indeed, network devices can become virtual ones that can be setup almost anywhere in the infrastructure. In general, this concept is known as *network virtualization* and even if it can be implemented by software-based solutions, the use of reconfigurable hardware enables the support of high-speed network processing as required today.

In this context, we consider the case of implementing a Layer 3 (L3) packet switch device with capabilities of forwarding packets between different virtual local area networks (VLANs). It is worth mentioning that this device is widely used to implement different networks which share the same infrastructure. In particular, the configuration referred as *Router on a Stick* can be useful to process all inter-VLAN traffic aggregated by a Layer 2 (L2) switch. This switch can group packets on an output (trunk) port by tagging them with VLAN identifiers ($VIDs$) at input (access) ports. Forwarding within VLANs (i.e., intra-VLAN traffic) is implemented by the switch itself by simply processing the VID tag and imposes no critical constraints. However, routing packets between VLANs (i.e., inter-VLAN traffic) demands further classification and modification of the tag header. This configuration is shown in Figure 5. Inter-VLAN traffic can only be forwarded to the correct VLAN by processing the IP destination address of the packet, which determines the destination VLAN where the packet is addressed to. Thus, the VID value needs to be modified based on this decision to let the L2 switch deliver the packet to its final host.

For the sake of simplicity in our analysis, we focus our discussion on the basic processing path for the *Router on a Stick* case. As just discussed, this path must implement two processes: classification and tagging. Incoming packets are first classified based on their IP destination address. To this end, a classifier function is implemented, which can handle a maximum number R of classification rules. Each rule associates one IP network address to one VID value. The classifier determines whether a given destination IP value can be associated to a IP network one. If it does, the packet is marked on a control field with the matched rule number. Next, a tagger funcion reads this control field and updates the VID value available on the packet header with that associated to the matched rule.

Our implementation can support different values of R aiming at scaling if the infrastructure requires to handle an increasing number of VLANs. As R

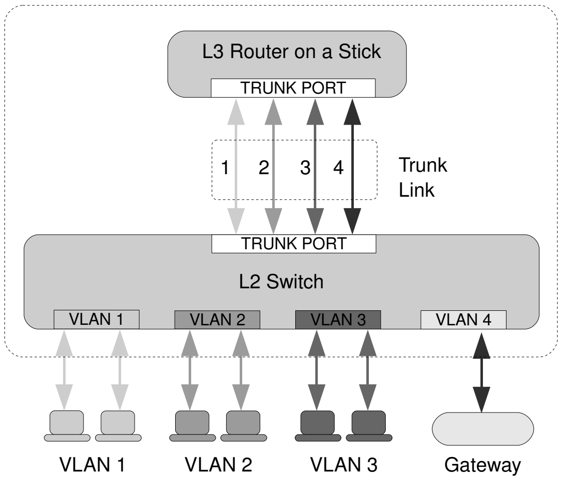

Fig. 3. Router on a stick configuration

increases, so does the complexity of the implementation. This is shown in Figure 4(a), where the usage of combinational elements on a Altera Stratix GXII FPGA device is reported when considering the case of $R = 4, 8, 16, 32, 64$. In all cases the implementation requires less than 1% of the available combinational elements; thus, enabling the integration of multiple instances on the same FPGA device. In general, the required combinational resources increase linearly with the number of rules ($R \geq 16$); however, for small values of R, combinational resources can be better optimized resulting in a lower demand. Other resources such as registers and memory blocks show little variations as R increases, so they can be considered almost constant for simplicity.

An increase on R also impact on the maximum frequency (F_{max}) that can be used to run the system. In our case, the architecture has a data path width (D_{width}) of 64 bits; thus, the resulting performance of the system in terms of processing bandwidth can be computed as $F_{max} \times D_{width}$. Figure 4(b) reports the maximum processing bandwidth in Gigabits per second (Gbps) units for different values of R. Note that reconfigurability enables the use of the most appropiate solution. Therefore, performance can always be increased to match the actual needs in terms of the maximum number of VLANs that must be supported. It is worth mentioning that reconfigurable hardware can easily support high-speed network processing at several Gbps, which is hard to achieve using only software-based solutions.

6 Conclusion

In this paper we dicussed the increasing relevance of FPGA technology in building network processing architectures. A general discussion was first introduced

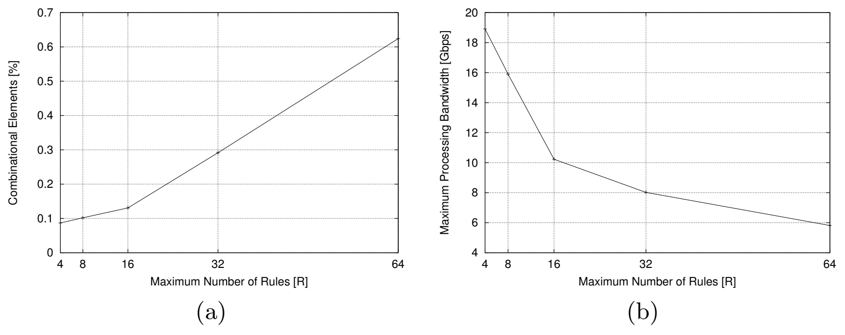

Fig. 4. Router on a stick implementation: a) Resource Usage, b) Maximum Performance

to highlight the main benefits of the FPGA technology. Next, the specific requirements for implementing network processing architectures were described and analyzed considered other technologies. Finally, a simple case study, based on our experience in the field, was discussed to illustrate the main drivers to consider FPGA as an interesting technology for network processing.

Acknowledgements This work was supported by Universidad Tecnologica Nacional FONCyT IP-PRH 2007 Posgraduate Grant Program and FONCyT PICT2007-534 Research Grant.

References

1. Hermsmeyer, C., Song, H., Schlenk, R., Gemelli, R., and Bunse, S.: Towards 100G packet processing: Challenges and technologies, Bell Labs Technical Journal, vol. 14., no. 2, 2009
2. Cisco White Paper: Approaching the Zettabyte Era, Cisco Systems, 2008
3. Leigh, K., Ranganathan, P., Sughlok, J.: Fabric Convergence Implications on Systems Architecture. In: IEEE 14th International Symposium on High Performance Computer Architecture HPCA 2008, pp.15-26. IEEE Press, New York (2008)
4. Kohler, E., Morris, R., Chen, B., Janotti, J., and Kaashoek, M.F: The Click Modular Router. ACM Transactions on Computer Systems, vol. 18, no. 3, pp. 263-297 (2000)
5. Schelle, G., Grunwald, D.: CUSP: A Modular Framework for High Speed Network applications on FPGAs, in: ACM/SIGDA 13th International Symposium on Field-Programmable Gate Arrays, pp. 246-257. ACM, New York (2005)
6. Nikander, P., Nyman, B., Rinta-aho, T., Sahasrabuddhe, S., Kempf, J.: Towards Software-defined Silicon: Experiences in Compiling Click to NetFPGA, in: Proceedings of the 1st European NetFPGA Developers Workshop (2010)
7. Brebner, G.: Single-chip gigabit mixed-version IP router on Virtex-II Pro, in: 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 35- 44. IEEE Press, New York (2002)

8. Shafer, J., Rixner, S.: RiceNIC: a reconfigurable network interface for experimental research and education, in: Proc. of the Workshop on Experimental Computer Science ExpCS 07, Part of ACM FCRC.ACM, San Diego (2007)
9. Gibb, G., Lockwood, J., Naous, J., Hartke, P., and McKeown, N.: NetFPGA - An Open Platform for Teaching How to Build Gigabit-rate Network Switches and Routers. IEEE Transactions on Education, vol. 51, no. 3, pp. 364-369 (2008)
10. Finochietto, J., Paz, S., Zerbini, C.: Hardware Primitives for Packet Flow Processing Architectures, in: VII Southern Conference on Programmable Logic (SPL2011), pp. 37-43. IEEE (2011)
11. Casado, M., Koponen, T., Moon, D., Schenker, S.: Rethinking Packet Forwarding Hardware, in: Proceedings of ACM Special Interest Group on Data Communications Workshop on Hot Topics in Networks (HotNets-VII), pp. 1-6. ACM, New York (2008)
12. Dobrescu, M., Egi, N., Argyraki, K., Chun, B., Fall, K., Iannaccone, G., Knies, a., Manesh, M., Ratnasamy, S.: Routebricks: Exploiting Parallelism To Scale Software Routers, in: 22nd ACM Symposium on Operating Systems Principles (SOSP). ACM, New York (2009)
13. Giladi, R.: Network Processors: Architecture, Programming, and Implementation. Morgan Kaufmann, Burlington (2008)
14. Jiang, W., Prasanna, V.: Large-Scale Wire-Speed Packet Classification on FPGAs, in: ACM/SIGDA International Symposium on Field Programmable Gate Arrays (FPGA'09), pp. 219-228. ACM, New York (2009)
15. Qi, Y., Fong, J., Jiang, W., Xu B., Li, J., Prasanna, V.: Multi-dimensional Packet Classification on FPGA: 100 Gbps and Beyond, in: International Conference on Field-Programmable Technology (FPT '10), Dec. 2010.
16. Taylor, D., Turner, J., Lockwood, J., Sproull, T., Parlour, D.: Scalable IP Lookup for Internet Routers. IEEE Journal on Selected Areas in Communications, vol. 21, no. 4, pp. 522-534 (2003)
17. Chao, J., Liu, B.: High Performance Switches and Routers. Wiley & Sons, New Jersey (2007)
18. Integrating 100-GbE Switching Solutions on 28-nm FPGAs. Altera White Paper, 2010